Why can I not start a new line before a brace group

This was split off from the Tcl 9.0 WishList, as it might be more generally informative.

GPS: May 20, 2004 - I would like to note that I wrote these questions/statements years ago. I've come to understand the beauty of Tcl's syntax rules. :)

GPS - I would like support for **more flexible indentation of curly braces** (Allman style indentation). Then I wouldn't have to use \ to get the curly braces the way that I want them.

RS - Would that be

```
if {$cond}
{
...
}
```

GPS - Yes

RS - This is fundamentally impossible in Tcl, since newline (or semicolon) terminate a command, and the *then* body is the second argument to the *if* command. The parser has no special syntax for *if*, *for*, *while*,... - and it should not have, even in 10.0!

LCS - Not impossible, it just requires some sideways thinking. For example, suppose the "if" command were to stack an internal boolean result and a "level". Then the "then" and "else" commands at the same level would check that boolean to see if they should execute their associated code blocks. The "then" command could unstack the boolean if it was true, leaving the "else" command with nothing to do, and would leave it for the "else" command if it was false.

```
if { $cond }
then {
}
else {
}
```

One could also imagine a "gimmee the next line" command that would allow a command to change the interpretation of the succeeding commands.

```
proc foo { a b c } {
  set d [ getline ]

  puts "d is \"$d\"
}

proc grill { a b c } {
  puts "hi there"
}
```

where:

```
foo 1 2 3
bar 4 5 6
grill 7 8 9
```

results in:

```
d is "bar 4 5 6"
hi there
```

Having grabbed the next line by fair means or foul, "foo" can now do whatever it jolly well likes with it. Likely we would need an "ungetline".

```
proc if { cond } {
  set then [getline]
  set else [getline]
  internal_if { [ lindex $else 0 ] != "else" } { ungetline $else; set else "" }
  internal_if { $cond } { eval [ lrange $then 1 end ] } {
    internal_if {$else != ""} { eval [ lrange $else 1 end ]}
  }
}
```

GPS - Why shouldn't the parser have a special syntax for proc, if, for, while, and foreach?

RS - The Tcl Way: all commands are treated equally: a list of words, the first being the command name, the others its arguments; commands separated by newlines or semicolons. Allowing unescaped newlines inside calls to proc..., but of course not for every command ;-), would make this quite a different language, more C-like, less Tcl-like. But I like Tcl the way it is!

GPS - But I don't like the way this aspect of Tcl is!

? - It seems as if what you really want is another language that is a bit like Tcl but different in a number of ways. Thank goodness the source code for Tcl is openly available so that when someone, like yourself, finds such a need, they can create their own specialized language that works however they desire.

DKF - Changing this is hard. This is because to do so requires the introduction of a proper parser (LL(k) or LALR(1)), and that conflicts with the flexibility of the language as it currently stands. Either you lose support for things like *-command* arguments to Tk widgets, or you build a parser that can be significantly extended at runtime (not just new

keywords that are patterned like existing ones, but wholly new constructs.) This is non-trivial to create, and even harder to actually use (Do you understand the difference between a shift-reduce and a reduce-reduce conflict, and appreciate why both are bad? Can you work out lookahead limits in your head?)

Take it from me, what you ask for is out of reach in Tcl.

GPS - I would like to note that I have changed my mind and don't currently feel that this aspect of Tcl should change.

NEM - You can do this style of writing with Tcl, if you do it within a command that defines its own syntax:

```
proc tclify {script} {
    regsub -all {\n\s*(\{.*)$} $script { \1} script
    uplevel 1 $script
}
```

And then you can do stuff like:

```
tclify {
   for {set a 1} {$a < 10} {incr a}
   {
       puts "a = $a"
   }
}
```

You have to call the command to make this work, but you can always override source etc to do it for you.

DKF - Cool suggestion. I'll have to remember that for the next time this topic comes up. :^)

jcw - For a hair-raising alternative, see an indentation syntax for Tcl ...

GPS - I would also like multiline comments, that don't require using \ at the end to continue, but this would of course violate The Tcl Way.

RS - For this I normally use the

```
if 0 {
        ...
}
```

construct, where you can brace away everything - except unbalanced braces ;-)

DKF - I just use Emacs's rectangular editing commands to put a comment character at the start of each line I want to comment out. So long as I comment out balanced blocks (the usual case in my coding style) I have no problems at all, ever.

AK - Don't forget the commands (un)comment-region. Mark a region, execute the command and emacs will remove/add the comment character to each line in the region. This will not work for modes with no comment-character defined. tcl mode defines #.

DKF - I never remember that one, but I use the rectangle commands extensively. *:^)*

rdt - What about something as stupid as:

```
proc \# {args} {}
```

and then you can say:

```
\# {
this is a long
and/or multi
line comment.
}
```

DKF: Just a note to say that both the [if 0 ...] and the [\# ...] versions are bytecode-efficient.

Category syntax